TL-Training: A Task-Feature-Based Framework for Training Large Language Models in Tool Use

Anonymous ACL submission

Abstract

Large language models (LLMs) achieve remarkable advancements by leveraging tools to interact with environments, a critical step toward generalized AI. However, the standard supervised fine-tuning (SFT) approach, which relies on large-scale datasets, often overlooks task-specific characteristics in tool use, leading to performance bottlenecks. To address this issue, we analyze three existing LLMs and uncover key insights: training data can inadvertently impede tool-use behavior, token importance is distributed unevenly, and errors in tool calls fall into a small set of categories. Building on these findings, we propose *TL-Training*, a task-feature-based framework that mitigates the effects of suboptimal training data, dynamically adjusts token weights to prioritize key tokens during SFT, and incorporates a robust reward mechanism tailored to error categories, optimized through proximal policy optimization. We validate TL-Training by training CodeLLaMA-2-7B and evaluating it on four open-source test sets. Our results demonstrate that the LLM trained by our method matches or surpasses both open- and closed-source LLMs in tool-use performance using only 1,217 training data points. Additionally, our method enhances robustness in noisy environments and improves general task performance, offering a scalable and efficient paradigm for tool-use training in LLMs.[1]

1 Introduction

Large language models (LLMs) (OpenAI, 2023; Touvron et al., 2023; Bai et al., 2023) excel in natural language understanding due to pre-training on extensive datasets (Chen et al., 2023; Ye et al., 2023). By incorporating tool-use capabilities, LLMs can extend beyond text generation to interact with the external environment, enabling tasks such as web searches and email management (Tang et al., 2023; Ye et al., 2025). Furthermore, these capabilities are essential for addressing real-world user needs and advancing the development of general-purpose AI (Xi et al., 2023).

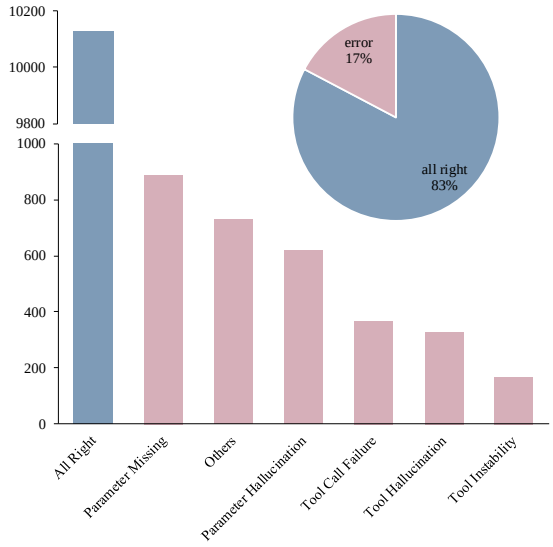

Figure 1: Error statistics for various tool calls in RoTLLaMA's training data.

Current approaches to training LLMs for tool use rely heavily on large-scale datasets generated from trajectories of interactions with tools (Qin et al., 2024; Zhuang et al., 2023). Standard supervised fine-tuning (SFT) is then applied to pre-trained models. While effective in some cases, these methods overlook key task-specific characteristics, leading to performance bottlenecks. For instance, ToolLLaMA-2-7B-v2 (Qin et al., 2024) achieves only 80% of GPT-4's performance on tool-use benchmarks (Ye et al., 2025; Wu et al., 2024), indicating room for significant improvement.

To fill this gap, we conduct an in-depth analysis of three tool-using LLMs, uncovering several key phenomena. Notably, over 17% of the training data for RoTLLaMA (Ye et al., 2024b)

[1] Code and data will be available upon acceptance of the paper.

Table 1: Proportion statistics of various error cases in RoTBench (Clean) for ToolLLaMA-2-7B-v2 and NexusRaven-13B-v2. 'First' indicates a mismatch in the initial token of the selected and correct tool names, while 'Prefix' denotes a shared prefix between them. 'Synonyms' captures instances where filled parameter values are synonymous with standard values.

Aspect	Error	ToolLLaMA	NexusRaven
Tool	First	53.33%	65.22%
	Prefix	46.67%	34.78%
Parameter	Redundancy	46.43%	33.33%
	Missing	57.14%	66.67%
	Hallucination	17.86%	6.67%
Content	Synonyms	73.68%	95.00%
	Others	31.58%	5.00%

contains tool-calling errors (Figure 1), primarily due to reliance on data from GPT series models, which are not entirely error-free with complex tools. Training on such flawed data can hinder model performance. Additionally, our tests of ToolLLaMA-2-7B-v2 and NexusRaven-13B-v2 (team, 2023) reveal that incorrect tool selections often share a common prefix with the correct ones (Table 1), and correcting initial erroneous tokens can lead to successful predictions. This suggests that certain tokens are more critical in tool selection. Moreover, the types of errors produced by tool calls are relatively limited (Figure 3), providing a foundation for targeted improvements across various error categories.

Based on these insights, we propose *TL-Training*, a task-feature-based framework for training LLMs in tool use. TL-Training mitigates the negative impact of training data by identifying erroneous interaction paths and excluding them from gradient updates. It prioritizes key tokens through adaptive weighting during SFT and incorporates tool feedback into a robust reward mechanism for reinforcement learning using the proximal policy optimization (PPO) (Schulman et al., 2017).

We validate our approach by training CodeLLaMA-2-7B (Rozière et al., 2023) on a curated dataset of 1,217 tool-call trajectories generated with GPT-4o. Evaluations on four open-source test sets demonstrate that the model trained with TL-Training matches or surpasses the tool-use performance of leading open- and closed-source LLMs, despite requiring significantly less training data. Additionally, TL-Training improves robustness in noisy environments and enhances general task performance.

In summary, our contributions are as follows: 1) We identify three key insights in tool use, including the impact of erroneous data, the uneven importance of tokens, and the constrained range of tool-calling error categories; 2) We propose TL-Training, a novel task-feature-based framework comprising of adverse effects mitigation, key tokens prioritization, and reinforcement learning for tool use; 3) We demonstrate the effectiveness of TL-Training by training CodeLLaMA-2-7B and achieving leading tool-use performance on multiple benchmarks with only 1,217 pieces of data; and 4) We show that TL-Training enhances both robustness to noisy data and general task performance, highlighting its potential for scalable tool-use training.

2 Preliminaries

Task Formulation Given a model $\mathcal{M}$, a user query q, and a collection of tools $\mathbb{T}$, the task of tool use requires $\mathcal{M}$ to iteratively select the appropriate tool $t_s \in \mathbb{T}$ at each step s, process its feedback o_s, and continue selecting subsequent tools t_{s+1} until the query is resolved and a final answer is obtained. Formally, this can be represented as $t_{s+1} = \mathcal{M}(\cdot|q, \mathbb{T}, t_{0..s}, o_{0..s})$. This task is distinct from traditional natural language processing tasks, as it requires the model to invoke tools repeatedly and interpret their feedback dynamically. Despite its importance, to the best of our knowledge, there has been no systematic examination of the intrinsic properties of tool use. Thus, we aims to fill this gap by conducting an in-depth analysis focusing on both the training data and model performance.

Data Analysis For our analysis, we use the training set from RoTLLaMA, which includes 12,247 filtered multi-turn tool-call trajectories generated by GPT-4. Illustrated in Figure 1, 17% of these trajectories contain various errors, indicating that even advanced models like GPT-4 encounter challenges with complex tools. These erroneous trajectories pose a challenge for models trained through SFT, as they inherit these error patterns during learning. This predisposition to incorrect tool invocation highlights the need for more robust training methods to mitigate error propagation and improve overall model performance.

Performance Analysis We evaluate the performance of ToolLLaMA-2-7B-v2 and NexusRaven-13B-v2, which are built on LLaMA-2-7B (Touvron

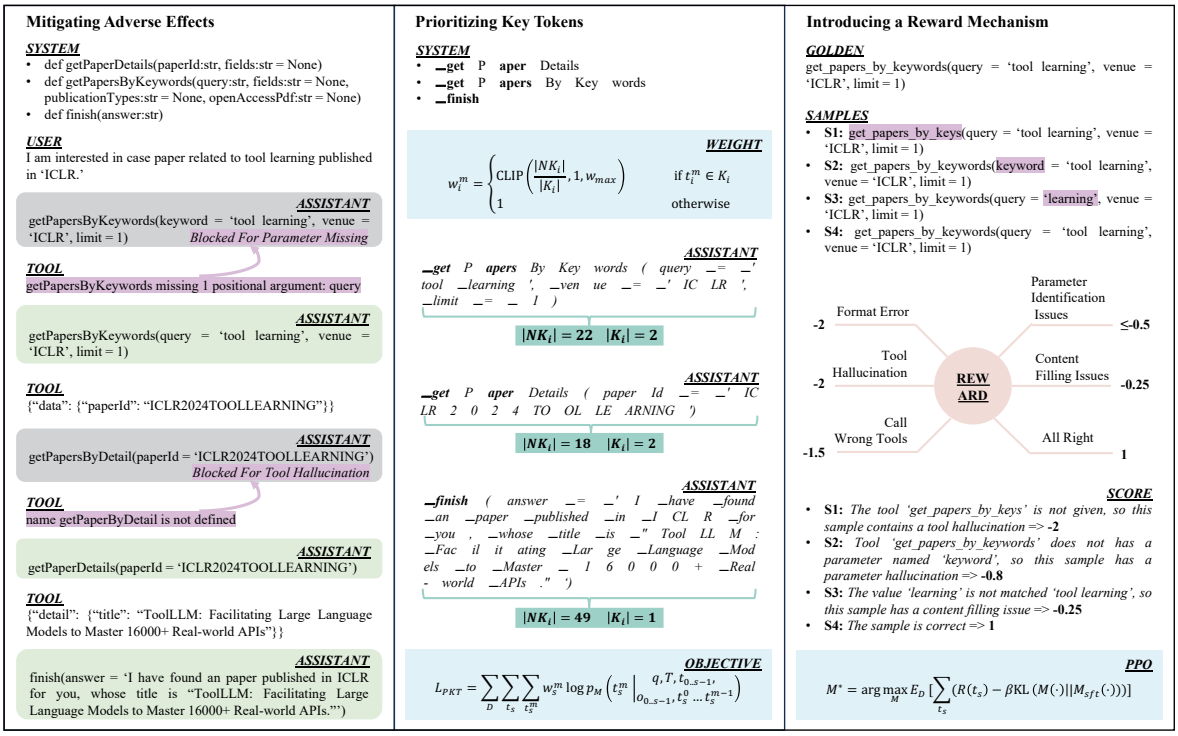

Figure 2: Framework of *TL-Training*. TL-Training comprises three main components: (**Left**) mitigating the adverse effects of suboptimal data by identifying erroneous interaction trajectories through tool feedback and blocking their gradient updates; (**Middle**) optimizing key tokens by dynamically adjusting token weights during the SFT process; and (**Right**) enhancing tool call performance through a reward mechanism tailored to tool invocation error types, using the PPO algorithm for reinforcement learning.

et al., 2023) and CodeLLaMA-2-13B (Rozière et al., 2023), respectively, as representative tool-using LLMs. Our evaluation uses RoTBench (Clean) (Ye et al., 2024b), a manually labeled dataset for the single-turn tool-use task with standardized answers. The analysis focuses on errors related to tool selection, parameter identification, and content filling, with results shown in Table 1. We observe that when the model selects the wrong tool, it often chooses one with a prefix similar to the correct tool. By manually correcting the first incorrectly predicted token, the model can generate the correct one, suggesting that certain tokens are crucial for task success. Additionally, errors in parameter identification and content generation highlight areas where further training is required.

3 Approaches

Building on the analysis in Section 2, we propose TL-Training, a novel training paradigm for LLMs in tool use. As shown in Figure 2, this paradigm incorporates three core techniques: mitigating the adverse effects of suboptimal data by preventing its back-propagation (Section 3.1), prioritizing key tokens using adaptive weight adjustments (Section 3.2), and implementing a reward mechanism tailored to tool invocation error categories to enable effective reinforcement learning (Section 3.3).[2]

3.1 Mitigating Adverse Effects

During the SFT stage, the objective is to align LLMs with the distribution of the training data. However, erroneous interaction paths in the data can negatively affect the model's decision-making, leading to an increased likelihood of incorrect tool calls. To address this, we design an automated process that identifies erroneous interaction paths and blocks their back-propagation, thereby reducing their harmful impact on the model.

Given a data sequence $(q, t_{0..s}, o_{0..s})$, we seek to identify the erroneous tool call trajectory $\mathbb{T}_e \subseteq \{t_0, t_1, \ldots, t_s\}$. Directly determining whether a specific t_i is correct is challenging. However, the feedback o_i generated after each tool call contains structured error-reporting information, as summarized in Figure 3.[3] We automate the

[2]Theoretical proofs of the effectiveness of our proposed approaches is provided in Appendix A.

[3]Specific examples can be found in Appendix E.

Figure 3: Types of errors encountered by LLMs during tool use and their corresponding feedback messages.

identification of incorrect calls by sequentially analyzing o_i to extract $\mathbb{T}_e$.

Once $\mathbb{T}_e$ is identified, we mitigate the impact of these erroneous interactions by blocking their back-propagation during training. This is achieved by modifying the loss function as follows:

$$\mathcal{L}_{MAE} = -\sum_{\mathbb{D}} \sum_{t_s \notin \mathbb{T}_e} \log p_M(t_s | q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}),$$

where $\mathbb{D}$ represents the entire training dataset.

3.2 Prioritizing Key Tokens

Based on the analysis in Section 2, and the insights from rows 1-2 of Table 1, we observe that the first token of a tool name, along with any subsequent token that shares a common prefix with other tool names, plays a more critical role in successful tool identification. As such, these tokens are more challenging for LLMs to generate correctly. However, standard SFT training maximizes the conditional probability of each token without distinction, treating all tokens as equally important. To address this limitation, we propose a scheme that adaptively adjusts the training weights of tokens according to their relative importance.

Given a data sequence $(q, t_{0..s}, o_{0..s})$, where each tool $t_i = (t_i^0, t_i^1, \ldots, t_i^{l_i})$ consists of l_i tokens, we categorize the tokens into two sets:

$$K_i = \{t_i^m \in t_i \mid t_i^m \text{ is a key token}\}$$

$$NK_i = \{t_i^m \in t_i \mid t_i^m \text{ is not a key token}\}$$

We then adjust the weights of K_i and NK_i based on their relative importance, allowing the model to focus more on the key tokens.

$$w_i^m = \begin{cases} \text{CLIP}\left(\frac{|NK_i|}{|K_i|}, 1, w_{\max}\right) & \text{if } t_i^m \in K_i \\ 1 & \text{otherwise} \end{cases}$$

Here, $w_{\max}$ is the maximum adjustment multiplier, and $\text{CLIP}(x, \min, \max)$ is used to constrain the adjustment factor to lie within the range $[\min, \max]$. The notation $|\cdot|$ represents the size of the set.[4]

With these computed weights, we prioritize key tokens during training with the following objective:

$$\mathcal{L}_{PKT} = -\sum_{\mathbb{D}} \sum_{t_s} \sum_{t_s^m} w_s^m \cdot \log p_M(t_s^m \mid q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}, t_s^0 \ldots t_s^{m-1}).$$

3.3 Introducing a Reward Mechanism

The three stages of tool use by LLMs are interdependent, where an error in any stage can lead to the failure of the entire tool invocation. Fortunately, the types of errors that arise are limited, enabling us to introduce a reward mechanism based on these specific errors. This allows us to apply reinforcement learning algorithms that help align the model more closely with human intent and enhance its tool-use proficiency. To achieve this, we define a set of reward functions tailored to the tool use task and employ the PPO algorithm to optimize the model's performance.

Given an LLM-generated tool call prediction t_i and its corresponding ground truth, we define the following reward function based on the quality of the LLM's tool use in various scenarios:

$$R(t_i) = \begin{cases} -2 & \text{if } t_i \text{ cannot be parsed} \\ -2 & \text{if } t_i \text{ contains tool hallucinations} \\ -1.5 & \text{if } t_i \text{ calls the wrong tool} \\ R_{\text{p}}(t_i) & \text{if } t_i \text{ has parameter issues} \\ -0.25 & \text{if } t_i \text{ has content filling issues} \\ 1 & \text{if } t_i \text{ is correct} \end{cases}$$

where $R_{\text{p}}(t_i)$ is defined as:

$$\begin{aligned} R_{\text{p}}(t_i) = &-0.8 \cdot \mathbb{I}(t_i \text{ has parameter hallucinations}) \\ &-0.5 \cdot \mathbb{I}(t_i \text{ has redundant parameters}) \\ &-0.5 \cdot \mathbb{I}(t_i \text{ has missing parameters}) \end{aligned}$$

where $\mathbb{I}(\cdot)$ represents the indicator function.

[4] Since K_i always includes at least the first token of the tool name, we avoid any risk of dividing by zero.

This reward function R addresses the different potential errors in LLM tool use, providing a structured scoring system to assess performance. Based on this, we apply the PPO algorithm, which iteratively optimizes the model's parameters to maximize these rewards as follows:

$$\mathcal{M}^* = \arg\max_{\mathcal{M}} \mathbb{E}_{\mathbb{D}}[\sum_{t_s}(R(t_s) - \beta \mathrm{KL}(\mathcal{M}(\cdot)||\mathcal{M}_{sft}(\cdot)))]$$

where β regulates deviation from the initial SFT model $\mathcal{M}_{sft}$. This approach enables the LLM to progressively refine its understanding and improve the accuracy of its tool usage over time.

4 Experimental Setup

4.1 Dataset

As shown in Table 2, to validate our approach, we construct a custom training set focused on multi-turn tool use and evaluate it using four publicly available test sets (i.e., ToolAlpaca (Tang et al., 2023), RoTBench (Ye et al., 2024b), BFCL-v3 (Patil et al., 2024), and ToolEyes).[5]

4.2 Baselines

We conduct a comprehensive comparison of ten LLMs from three different categories. These include **ToolLLaMA-2-7B** and **NexusRaven-2-13B** as tool-use LLMs; **ChatGLM-4-chat-9B** (Zeng et al., 2024), **Qwen-2-Instruct-7B** (Yang et al., 2024a), **LLaMA-3.1-Instruct-8B** (Team, 2024), and **Qwen-2.5-Instruct-7B** (Yang et al., 2024b) as open-source LLMs; **GPT-3.5-turbo**, **GPT-4o**, and **GPT-4-turbo** as closed-source LLMs; and **TL-CodeLLaMA-2**, developed from CodeLLaMA-2-7B with the custom dataset.[6]

4.3 Metrics

For **single-turn** tool use, where the original dataset provides a standard answer, we follow Ye et al. (2024b) and assess the model's performance across three key areas: 1) **Tool Selection (TS)** measures the model's accuracy in selecting the tool specified by the standard answer; 2) **Parameter Identification (PI)** evaluates the model's ability to correctly select the tool and identify the relevant parameters required for invocation; and 3) **Content Filling (CF)** assesses the model's capacity to complete the single-turn tool invocation, including selecting the correct tool, identifying relevant parameters, and filling in the appropriate values.

For **multi-turn** tool use, where no standardized interaction path exists, we adapt the methods of Qin et al. (2024) and Ye et al. (2025), and assess performance based on following metrics: 1) **Documentation Understanding Error (DE)** represents the percentage of errors resulting from the model's failure to interpret the tool documentation, encompassing tool hallucinations, parameter hallucinations, and missing necessary parameters; 2) **Tool Call Error (CE)** denotes the proportion of errors arising from incorrect tool invocation, covering all error types except those classified as DE; and 3) **Valid Answers (VA)** evaluates the percentage of instances where the model delivers valid responses within nine turns.

Table 2: Statics of datasets used. '# Number' represents the number of data in the dataset. 'Type' represents the type of tool use.

Split	Dataset	# Number	Type
Train	Self-Construct	1217	Multi-Turn
Test	ToolAlpaca	114	Single-Turn
	RoTBench	105	Single-Turn
	BFCL-v3	239	Single-Turn
	ToolEyes	382	Multi-Turn

4.4 Implementation Details

In the **SFT** stage, we use 1,217 constructed data samples, applying both the MAE and PKT strategies. We employ the AdamW optimizer (Loshchilov and Hutter, 2019) with cosine scheduling, setting the learning rate to 1e-6, a warmup rate of 0.01, and a batch size of 4, training for a total of 1 epoch. For the PKT strategy, w_{max} is set to 9. In the **RL** stage, we filter 1,194 entries from the constructed data and apply PPO with the reward function described in Section 3.3. The actor learning rate is set to 2e-6, the critic learning rate to 1e-6, and the batch size to 8, training for a total of 3 epochs. For **testing**, we use the official prompt template for tool invocation and apply greedy search.[7]

5 Experiments

5.1 Main Results

We evaluate the performance of various LLMs on three single-turn tool-use test sets and one multi-

[5]Details of the datasets can be found in Appendix B

[6]Details of baselines can be found in Appendix C.

[7]Templates for each LLM are provided in Appendix D.

Table 3: Performance of various LLMs on single-turn test sets. 'Avg.' represents the average performance across all LLMs. Individual LLM performances are color-coded for clarity: teal highlights better-than-average performance, while purple indicates below-average performance. Darker shades signify greater deviations from the average. The best performance in each column is indicated in **bold**.

Models	Size	ToolAlpaca			RoTBench			BFCL-v3		
		TS (↑)	PI (↑)	CF (↑)	TS (↑)	PI (↑)	CF (↑)	TS (↑)	PI (↑)	CF (↑)
Avg.		*80.63*	*65.09*	*42.72*	*74.10*	*49.90*	*35.43*	*93.13*	*89.88*	*74.17*
ToolLLaMA-2	7B	75.56	61.40	37.72	70.48	43.81	25.71	87.08	83.75	56.67
NexusRaven-2	13B	82.46	48.25	37.72	70.48	56.19	37.14	97.08	94.17	75.83
ChatGLM-4-chat	9B	73.68	68.42	38.60	67.62	53.33	37.14	95.42	93.33	86.67
Qwen-2-Instruct	7B	86.84	68.42	43.86	74.29	47.62	35.24	98.33	95.42	85.00
LLaMA-3.1-Instruct	8B	84.21	59.65	42.98	62.86	17.14	8.57	63.75	59.58	34.58
Qwen-2.5-Instruct	7B	**92.11**	68.42	44.74	80.00	32.38	15.24	**100.00**	95.42	87.92
GPT-3.5-turbo	-	72.81	54.39	39.47	74.29	61.90	48.57	99.17	**95.83**	75.83
GPT-4o	-	76.32	70.18	42.11	74.29	62.86	50.48	96.67	93.75	74.58
GPT-4-turbo	-	74.56	73.68	42.11	82.86	**69.52**	**53.33**	97.50	93.75	76.25
TL-CodeLLaMA-2	7B	87.72	**78.07**	**57.89**	**83.81**	54.29	42.86	96.25	93.75	**88.33**

Table 4: Performance on the multi-turn test set.

Models	ToolEyes		
	DE (↓)	CE (↓)	VA (↑)
Avg.	*3.91*	*12.46*	*65.56*
ToolLLaMA-2	21.00	36.62	52.36
ChatGLM-4-chat	0.17	32.45	43.45
Qwen-2-Instruct	0.78	6.71	70.16
LLaMA-3.1-Instruct	4.80	**3.76**	4.71
Qwen-2.5-Instruct	4.78	4.60	74.08
GPT-3.5-turbo	2.36	10.73	89.79
GPT-4o	**0.12**	4.64	87.43
GPT-4-turbo	0.34	7.83	**90.31**
TL-CodeLLaMA-2	0.82	4.84	77.75

turn tool-use test set, with the results summarized in Table 3 and Table 4.[8] Despite using the smallest model size and the least amount of training data, our approach achieves results comparable to the best-performing models. These findings demonstrate the potential for smaller, efficient models to excel in the tool use task, making advanced capabilities more accessible for resource-constrained environments.

Single-Turn Evaluation Results from three single-turn tool-use test sets demonstrate that TL-CodeLLaMA-2, with only 7B parameters and 1,217 training examples, surpasses all other open-source LLMs in overall task completion (i.e., CF). Remarkably, on the ToolAlpaca and BFCL-v3 datasets, TL-CodeLLaMA-2 outperforms GPT-4-turbo, the top-performing GPT family model, by an impressive 15.78% and 12.08%, respectively. Most notably, TL-CodeLLaMA-2 is the *only* model that consistently exceeds the average performance across all three aspects of every dataset evaluated, highlighting the effectiveness of our approach in enhancing single-turn tool usage capabilities.

Multi-Turn Evaluation In the multi-turn test set, our approach significantly enhances the ability of LLMs to handle the tool use task. TL-CodeLLaMA-2 achieves an total error rate of just 5.64% on the test set, second only to GPT-4o, which had the lowest error rate at 4.76%, and ahead of Qwen-2-Instruct at 7.49%. Additionally, TL-CodeLLaMA-2 maintaines a low error rate while achieving a high effective response rate, outperforming all other open-source models. In contrast, LLaMA-3.1-Instruct-8B frequently fails to provide a valid direct answer, effectively rendering it incapable of completing the task. These results highlight that our trained model effectively uses tools in multi-turn settings to solve complex user queries.

5.2 Ablation Studies

To assess the individual contributions of the three components in our design that enhance LLMs' tool-use capabilities, we conduct ablation studies,

[8] Since NexusRaven-2-13B does not receive tool feedback during interactions, it is evaluated only on the single-turn tool-use datasets.

Table 5: The ablation studies of the three components of our method, with better results than **Standard SFT** labeled in teal, poorer results labeled in purple. Darker colors indicate larger gaps.

Models	ToolAlpaca			RoTBench			ToolEyes		
	TS (↑)	PI (↑)	CF (↑)	TS (↑)	PI (↑)	CF (↑)	DE (↓)	CE (↓)	VA (↑)
Standard SFT (*w/ None*)	*74.56*	*70.18*	*42.98*	*76.19*	*55.24*	*38.10*	*0.72*	*9.08*	*59.32*
w/ MAE	73.68	64.91	43.68	78.10	57.14	40.95	1.22	5.53	68.85
w/ PKT	77.78	71.72	43.86	82.86	63.81	48.57	1.07	7.52	63.61
w/ IRM	88.60	84.21	57.02	79.05	59.05	44.76	0.65	6.80	57.33
w/ MAE & PKT	73.68	64.91	43.86	80.95	56.19	40.00	0.71	10.07	75.13
w/ MAE & IRM	87.72	78.95	57.89	82.86	56.19	45.71	0.96	5.03	71.20
w/ PKT & IRM	86.84	79.82	57.02	82.86	63.81	48.57	0.88	7.11	65.18
w/ All (TL-CodeLLaMA-2)	87.72	78.07	57.89	83.81	54.29	42.86	0.82	4.84	77.75

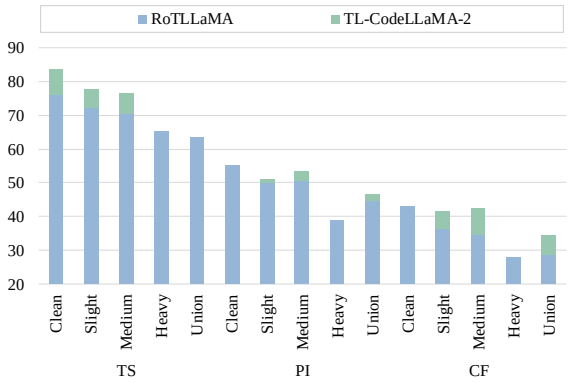

Figure 4: Performance comparison of RoTLLaMA and TL-CodeLLaMA-2 in different noise environments. RoTLLaMA's results are from Ye et al. (2024b).

comparing model performance across various scenarios. The results are shown in Table 5.

When compared to standard SFT without additional techniques (i.e., w/ None), masking erroneous interaction paths during training (e.g., w/ MAE) reduces the model's overall error rate in multi-turn tool use by nearly one-third and increases effective responses by 9.53%. This suggests that removing these erroneous paths prevents LLMs from learning incorrect tool-use patterns. However, since such errors are rare in single-turn tool use, the improvement in that case is less pronounced and requires additional techniques for further gains (e.g., w/ MAE & IRM). Similarly, optimizing the weights of key tokens during training (e.g., w/ PKT) enhances the model's ability to differentiate between similar tools, improving tool selection accuracy and overall performance. Furthermore, reinforcement learning with our proposed reward function (e.g., w/ IRM) further improves performance across all stages by dynamically optimizing the entire tool-use process, addressing diverse errors encountered during tool use. Finally, TL-CodeLLaMA-2 (i.e., w/ All), which integrates all three strategies, maximizes their combined advantages, significantly improving LLM performance in both single-turn and multi-turn tool use with only 1,217 data points, demonstrating the effectiveness of our approach.

6 Further Studies

6.1 Robustness Improvement

In real-world environments, tools often contain various types of noise, and LLMs must be robust in their tool use to effectively meet user needs across different situations. RoTBench provides five tool-use test environments with varying noise levels, designed to evaluate whether LLMs can accurately understand the functions and properties of different tools and execute effective invocations. We compare the performance of TL-CodeLLaMA-2 and RoTLLaMA across these five noisy environments, as shown in Figure 4. While RoTLLaMA has been optimized for such environments through targeted noise augmentation, TL-CodeLLaMA-2, without specific optimizations, matches or exceeds RoTLLaMA's performance in all aspects. This suggests that our approach allows the model to focus on the core functionality of external tools without being hindered by noise, making it more adaptable to real-world scenarios.

6.2 General Performance

The strong performance of LLMs is largely attributed to their extensive world knowledge and generalizability acquired during pre-training (Ye et al., 2023). However, fine-tuning on domain-specific tasks can sometimes compromise this generalizability (Yang et al., 2024c; Ghosh et al.,

2024). To assess whether TL-CodeLLaMA-2's general-purpose capabilities are affected by its exclusive training on tool-use data, we evaluate its performance on three general test sets: MMLU (knowledge) (Hendrycks et al., 2021), GSM8K (math) (Cobbe et al., 2021), and HumanEval (code) (Chen et al., 2021), comparing it to CodeLLaMA-2-7B. As shown in Figure 5, despite being fine-tuned solely for the tool-use task, TL-CodeLLaMA-2 retains its original task performance and even shows slight improvements in math and coding abilities. This is because our method requires only a small amount of training data, resulting in minimal changes to the model's original parameters, thus preserving its knowledge base (Ren et al., 2024; Wang et al., 2024; Ye et al., 2024c). Furthermore, the enhancement in tool-use ability appears to improve the model's reasoning capacity, contributing to better performance in math and coding tasks. These findings further underscore the broad applicability of our approach.

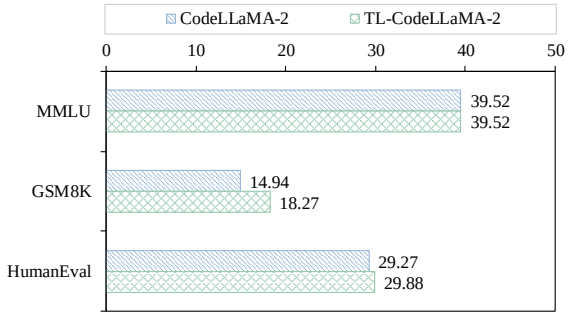

Figure 5: Performance comparison of CodeLLaMA-2 and TL-CodeLLaMA-2 across various general tasks.

7 Related Works

Training LLMs for Tool Use LLMs capable of utilizing external tools significantly enhance their ability to interact with dynamic environments and address user needs (Qin et al., 2023). However, the diversity and complexity of real-world tools present significant challenges in training such models. Existing methods, such as SFT, rely on the generation of extensive datasets of tool interactions (Song et al., 2023; Tang et al., 2023), enabling models to learn tool functionalities, invoke appropriate tools, and process feedback. While effective, these methods are resource-intensive due to the large-scale data construction involved. To overcome these challenges, some studies have proposed encoding tool names as special tokens directly integrated into model training, embedding tool-specific knowledge into the model (Hao et al., 2023). This approach has shown promise for existing tools but remains limited in its ability to adapt to newly introduced tools. Building on these findings, our work introduces a novel training paradigm for tool-use LLMs, addressing both efficiency and adaptability. By leveraging a compact dataset of 1,217 data points and incorporating three task-specific components, our approach achieves state-of-the-art performance while significantly reducing data requirements.

Evaluating LLMs in Tool Use Evaluating LLMs' tool-use capabilities is essential for understanding their effectiveness in diverse scenarios. A common evaluation method involves comparing predicted outputs with standard answers from a single turn of tool use (Chen et al., 2024). However, in multi-turn interactions, the variability in invocation processes complicates the definition of a single standard path. To address this, evaluations increasingly consider multiple dimensions of tool-use processes and outcomes (Ye et al., 2025). Beyond tool-use performance, researchers have also investigated robustness and safety in practical scenarios (Ye et al., 2024b,a), which provide insights into how LLMs manage edge cases and avoid harmful outputs. In this paper, we evaluate LLMs across single-turn and multi-turn tool use to provide a more comprehensive assessment. Additionally, we analyze robustness to further demonstrate the superiority of our approach.

8 Conclusion

In this paper, we introduce TL-Training, a novel paradigm for training LLMs specifically for tool use. Our approach mitigates the impact of erroneous interaction data, adaptively adjusts token weights, and introduces a reward mechanism tailed for tool use to facilitate PPO-based reinforcement learning. This methodology not only enhances LLMs' tool-use capabilities but also improves their robustness in noisy environments, all while preserving strong general performance across a range of tasks. Our findings demonstrate the effectiveness of TL-Training in addressing real-world challenges in tool use, offering a promising direction for future research in improving LLM interaction capabilities and adaptability.

Limitations

While we propose a novel paradigm for training LLMs in tool use, our work still has a few limitations. First, we do not construct large-scale training data. However, despite using only 1,217 data samples, our results show that we match or even surpass the best current tool-use performance, highlighting the strengths of our approach. Second, we design a reward function based on tool feedback for tool use directly, without training a separate reward model. Nonetheless, we experimentally demonstrate the effectiveness of our reward function. In future work, we plan to explore training a reward model specifically for tool learning to further improve model performance.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *CoRR*, abs/2309.16609.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Xuanting Chen, Junjie Ye, Can Zu, Nuo Xu, Rui Zheng, Minlong Peng, Jie Zhou, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. How robust is GPT-3.5 to predecessors? A comprehensive study on language understanding tasks. *CoRR*, abs/2303.00293.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. *Preprint*, arXiv:2312.14033.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Sreyan Ghosh, Chandra Kiran Reddy Evuru, Sonal Kumar, Ramaneswaran S., Deepali Aneja, Zeyu Jin, Ramani Duraiswami, and Dinesh Manocha. 2024. A closer look at the limitations of instruction tuning. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *CoRR*, abs/2305.11554.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, and 22 others. 2023. Tool learning with foundation models. *CoRR*, abs/2304.08354.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Mengjie Ren, Boxi Cao, Hongyu Lin, Cao Liu, Xianpei Han, Ke Zeng, Guanglu Wan, Xunliang Cai, and Le Sun. 2024. Learning or self-aligning? rethinking instruction fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 6090–6105. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *CoRR*, abs/2306.06624.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *CoRR*, abs/2306.05301.

Meta Team. 2024. Introducing llama 3.1: Our most capable models to date.

Nexusflow.ai team. 2023. Nexusraven-v2: Surpassing gpt-4 for zero-shot function calling.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Mengru Wang, Yunzhi Yao, Ziwen Xu, Shuofei Qiao, Shumin Deng, Peng Wang, Xiang Chen, Jia-Chen Gu, Yong Jiang, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2024. Knowledge mechanisms in large language models: A survey and perspective. *CoRR*, abs/2407.15017.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. *CoRR*, abs/2405.08355.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 10 others. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024a. Qwen2 technical report. *CoRR*, abs/2407.10671.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024b. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. 2024c. Self-distillation bridges distribution gap in language model fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 1028–1043. Association for Computational Linguistics.

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. A comprehensive capability analysis of GPT-3 and GPT-3.5 series models. *CoRR*, abs/2303.10420.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Tao Ji, Qi Zhang, Tao Gui, and Xuanjing Huang. 2025. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 156–187. Association for Computational Linguistics.

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024a. Toolsword: Unveiling safety issues of large language models in tool learning across three stages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2181–2211. Association for Computational Linguistics.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024b. Rotbench: A multi-level benchmark for evaluating the robustness of large language models in tool learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 313–333. Association for Computational Linguistics.

Junjie Ye, Yuming Yang, Qi Zhang, Tao Gui, Xuanjing Huang, Peng Wang, Zhongchao Shi, and Jianping Fan. 2024c. Empirical insights on fine-tuning large language models for question-answering. *CoRR*, abs/2409.15825.

Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, and 36 others. 2024. Chatglm: A family of large language models from GLM-130B to GLM-4 all tools. *CoRR*, abs/2406.12793.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2023. Toolchain*: Efficient action

space navigation in large language models with a* search. *CoRR*, abs/2310.13227.

A Theorems and Proofs

In this paper, we propose $\mathcal{L}_{MAE}$ and $\mathcal{L}_{PKT}$, aimed at enhancing the model's ability to utilize tools during the SFT stage. Additionally, we provide theoretical explanations of the effectiveness of these strategies.

Theorem A.1. *During the SFT stage for LLM in tool use, gradient updates resulting from incorrect interaction paths in the training data can adversely impact the model's ability to choose the appropriate tool.*

Proof. Let $\mathcal{M}$ be an LLM that interacts with a set of tools $\mathbb{T}$ to answer the user query q. At each step s, the model selects a tool $t_s \in \mathbb{T}$ based on the query and the history of tool selections and feedback:

$$t_{s+1} = \mathcal{M}(\cdot \mid q, \mathbb{T}, t_{0..s}, o_{0..s}),$$

where o_s is the feedback received after calling tool t_s.

Consider a dataset $\mathbb{D}$ comprising interaction sequences $(q, t_{0..s}, o_{0..s})$, which includes both correct and erroneous tool calls. Let $\mathbb{T}_e \subseteq \{t_0, t_1, \ldots, t_s\}$ denote the set of erroneous tool calls identified via analysis of feedback o_s.

The standard loss function during SFT aims to maximize the likelihood of the model's tool selections over the entire dataset:

$$\mathcal{L} = -\sum_{\mathbb{D}} \sum_{t_s} \log p_{\mathcal{M}}(t_s \mid q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}).$$

The gradient of this loss with respect to the model parameters θ is:

$$\nabla_\theta \mathcal{L} = -\sum_{\mathbb{D}} \sum_{t_s} \nabla_\theta \log p_{\mathcal{M}}(t_s \mid q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}).$$

This gradient comprises contributions from both correct and erroneous tool calls. The gradient component arising from erroneous tool calls is:

$$G_{\text{error}} = -\sum_{\mathbb{D}} \sum_{t_s \in \mathbb{T}_e} \nabla_\theta \log p_{\mathcal{M}}(t_s \mid q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}).$$

These gradients encourage the model to replicate erroneous tool selections, thereby misguiding its learning process. Specifically, they can increase the likelihood of the model making incorrect tool calls in future interactions, which negatively impacts its performance. By including erroneous tool calls in the gradient updates, the model parameters θ are adjusted in directions that do not align with optimal decision-making. This is detrimental because it interferes with the model's ability to learn the correct sequence of tool selections that effectively resolve user queries. Therefore, errors in the training data introduce gradient updates that adversely affect the model's performance. □

To mitigate this effect, we propose to modify the loss function to exclude erroneous tool calls from back-propagation:

$$\mathcal{L}_{MAE} = -\sum_{\mathbb{D}} \sum_{t_s \notin \mathbb{T}_e} \log p_{\mathcal{M}}(t_s \mid q, \mathbb{T}, t_{0..s-1}, o_{0..s-1}).$$

By omitting the erroneous tool calls from the loss computation, their associated gradients are not used to update the model parameters. This reduces the harmful impact of errors in the training data on the model's performance.

Theorem A.2. *During the gradient update process in SFT, assigning higher weights to key tokens prioritizes their contribution to the loss function, enabling the model to focus more on these tokens and fit them better.*

Proof. Let $\mathcal{M}$ be an LLM interacting with a set of tools $\mathbb{T}$ to answer the user query q. Each tool t_s used at step s is a sequence of tokens:

$$t_s = (t_s^0, t_s^1, \ldots, t_s^{l_s}),$$

where l_s is the length of the token sequence for tool t_s.

For each tool t_s, we categorize its tokens into two sets:

$$K_s = \{t_s^m \in t_s \mid t_s^m \text{ is a key token}\},$$

$$NK_s = \{t_s^m \in t_s \mid t_s^m \text{ is not a key token}\}.$$

We assign weights w_s^m to each token t_s^m as follows:

$$w_s^m = \begin{cases} \text{CLIP}\left(\frac{|NK_s|}{|K_s|},\ 1,\ w_{\text{max}}\right) & \text{if } t_s^m \in K_s, \\ 1 & \text{if } t_s^m \in NK_s, \end{cases}$$

where $|K_s|$ and $|NK_s|$ denote the number of key and non-key tokens in t_s, respectively, w_{max} is the maximum adjustment multiplier, and CLIP$(x,\ \text{min},\ \text{max})$ constrains x to the interval $[\text{min},\ \text{max}]$.

The modified loss function that prioritizes key tokens is:

$$\mathcal{L}_{PKT} = -\sum_{\mathbb{D}} \sum_{t_s} \sum_{t_s^m} w_s^m \cdot \log p_{\mathcal{M}}\left(t_s^m \mid q,\ \mathbb{T},\ t_{0..s-1},\ o_{0..s-1},\ t_s^0 \ldots t_s^{m-1}\right),$$

where $p_{\mathcal{M}}$ is the probability assigned by the model to token t_s^m, given the context.

The gradient of the loss with respect to the model parameters θ is:

$$\nabla_\theta \mathcal{L}_{PKT} = -\sum_{\mathbb{D}} \sum_{t_s} \sum_{t_s^m} w_s^m \cdot \nabla_\theta \log p_{\mathcal{M}}\left(t_s^m \mid q,\ \mathbb{T},\ t_{0..s-1},\ o_{0..s-1},\ t_s^0 \ldots t_s^{m-1}\right).$$

Tokens with higher weights w_s^m contribute more to the gradient:

$$\| w_s^m \cdot \nabla_\theta \log p_{\mathcal{M}}\left(t_s^m \mid q,\ \mathbb{T},\ t_{0..s-1},\ o_{0..s-1},\ t_s^0 \ldots t_s^{m-1}\right) \| \propto w_s^m.$$

Assuming $w_s^k > w_s^n$ for key token t_s^k and non-key token t_s^n, the gradient contribution from t_s^k is larger:

$$\| w_s^k \cdot \nabla_\theta \log p_{\mathcal{M}}\left(t_s^k \mid \cdots\right) \| > \| w_s^n \cdot \nabla_\theta \log p_{\mathcal{M}}\left(t_s^n \mid \cdots\right) \|.$$

During gradient descent, the parameter updates prioritize reducing the loss associated with higher-weighted (key) tokens:

$$\theta' = \theta - \eta \nabla_\theta \mathcal{L}_{PKT}.$$

As a result, the model adjusts its parameters more significantly to fit the key tokens, improving its ability to generate them correctly. This leads to better fitting of tokens with higher weights. Therefore, assigning higher weights to key tokens during gradient updates enhances the model's performance on these tokens. □

B Details of Datasets

As shown in Table 2, we construct a custom training set focused on multi-turn tool use and evaluate it using four publicly available test sets.

Training Data To train the LLMs using our method, we first construct a training dataset. Since ToolEyes provides a comprehensive set of invocable tools, we use it as a foundation to artificially create 1,217 relevant user requirements. GPT-4o is then employed to interact with these tools and generate the corresponding tool usage trajectories, which form our training set.

While previous studies often construct over 100,000 data points for training (Qin et al., 2024), we deliberately limit our dataset size. Our main goal is to validate the effectiveness of our approach rather than to scale data volume. Surprisingly, the experimental results in Section 5 show that training on just 1,217 data points using our method matches or even exceeds the performance of leading LLMs.

Test Sets To comprehensively evaluate LLM tool-use performance, we use four open-source tool usage test sets. ToolAlpaca (eval_real), RoTBench (Clean), and BFCL-v3 (executable) are selected for single-turn tool use evaluations, while ToolEyes is used for multi-turn tool use assessment.

C Details of Baselines

In this paper, we select nine existing LLMs from three different sources for a comprehensive comparison with our tool-use LLMs.

Tool-Use LLMs **ToolLLaMA-2-7B** and **NexusRaven-2-13B** are prominent tool-use LLMs, built on LLaMA-2-7B and CodeLLaMA-2-13B, respectively. These models are trained on a large volume of tool-use data, enhancing their ability to interact with tools. For example, ToolLLaMA-2-7B is trained on over 120,000 data points covering more than 16,000 tools using standard SFT, significantly boosting its tool-use capabilities.

Open-Source LLMs Among the existing open-source, general-purpose LLMs, **ChatGLM-4-chat-9B**, **Qwen-2-Instruct-7B**, **LLaMA-3.1-Instruct-8B** and **Qwen-2.5-Instruct-7B** have been specifically optimized for tool use, enabling them to interact with various tools to fulfill user needs.

Closed-Source LLMs The GPT family represents some of the most advanced LLMs, demonstrating strong performance not only in general-purpose tasks but also in tool use, with notable generalization capabilities. For this study, we select **GPT-3.5-turbo**, **GPT-4o**, and **GPT-4-turbo** as leading representatives of the GPT series for comparison.

Our Model We apply the TL-Training paradigm to CodeLLaMA-2-7B, using the custom dataset of 1,217 examples, to develop **TL-CodeLLaMA-2**, a specialized tool-use LLM. Compared to the other models in this study, TL-CodeLLaMA-2 is the smallest and trained on the least amount of data.

D Prompt Template

We use the official prompt of each LLM for tool use, which are provided from Table 6 to Table 13.

Table 6: An example for tool use of ToolLLaMA-2-7B.

System:
You are AutoGPT, you can use many tools(functions) to do the following task.
First I will give you the task description, and your task start.
At each step, you need to give your thought to analyze the status now and what to do next, with a function call to actually excute your step. Your output should follow this format:
Thought:
Action
Action Input:

After the call, you will get the call result, and you are now in a new state.
Then you will analyze your status now, then decide what to do next...
After many (Thought-call) pairs, you finally perform the task, then you can give your finial answer.
Remember:
1.the state change is irreversible, you can't go back to one of the former state, if you want to restart the task, say Ï give up and restart.
2.All the thought is short, at most in 5 sentence.
3.You can do more then one trys, so if your plan is to continusly try some conditions, you can do one of the conditions per try.
Let's Begin!
Task description: You should use functions to help handle the real time user querys. Remember:
1.ALWAYS call Finish function at the end of the task. And the final answer should contain enough information to show to the user,If you can't handle the task, or you find that function calls always fail(the function is not valid now), use function Finish->give_up_and_restart.
2.Do not use origin tool names, use only subfunctions' names.

Specifically, you have access to the following APIs: [{"type": "function", "function": {"name": "random_advice", "description": "Returns a random advice slip as a slip object.", "parameters": {"type": "object", "properties": {}, "required": []}}}]
User: Can you fetch some random advice for me?
Assistant:

Table 7: An example for tool use of NexusRaven-2-13B.

Function:
def random_advice():
"""
Returns a random advice slip as a slip object.

Args:

Returns:
string: The feedback from the tool.
"""

User Query: Can you fetch some random advice for me?<human_end>

Table 8: An example for tool use of ChatGLM-4-chat-9B.

< |system| >
你是一个名为 ChatGLM 的人工智能助手。你是基于智谱AI训练的语言模型 GLM-4 模型开发的，你的任务是针对用户的问题和要求提供适当的答复和支持。

可用工具

random_advice

{“name": “random_advice", “description": “Returns a random advice slip as a slip object.", “parameters": {“type": “object", “properties": {}, “required": []}}
在调用上述函数时，请使用 Json 格式表示调用的参数。< |user| >
Can you fetch some random advice for me?< |assistant| >

Table 9: An example for tool use of Qwen-2-Instruct-7B.

< |im_start| >system
You are a helpful assistant.< |im_end| >< |im_start| >user
Answer the following questions as best you can. You have access to the following tools:

random_advice: Call this tool to interact with the random_advice API. What is the random_advice API useful for? Returns a random advice slip as a slip object. Parameters: [] Format the arguments as a JSON object.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [random_advice]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can be repeated zero or more times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: Can you fetch some random advice for me?< |im_start| >assistant

Table 10: An example for tool use of LLaMA-3.1-Instruct-8B.

< |begin_of_text| >< |start_header_id| >system< |end_header_id| >

Environment: ipython
Cutting Knowledge Date: December 2023
Today Date: 26 Jul 2024

< |eot_id| >< |start_header_id| >user< |end_header_id| >

Given the following functions, please respond with a JSON for a function call with its proper arguments that best answers the given prompt.

Respond in the format "name": function name, "parameters": dictionary of argument name and its value.Do not use variables.

{
"function": {
"description": "Returns a random advice slip as a slip object.",
"name": "random_advice",
"parameters": {
"properties": {},
"required": [],
"type": "object"
}
},
"type": "function"
}

Can you fetch some random advice for me?< |eot_id| >< |start_header_id| >assistant< |end_header_id| >

Table 11: An example for tool use of Qwen-2.5-Instruct-7B.

< |im_start| >system
You are Qwen, created by Alibaba Cloud. You are a helpful assistant.

Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{“function”: {“description”: “Returns a random advice slip as a slip object.”, “name”: “random_advice”, “parameters”: {“properties”: {}, “required”: [], “type”: “object”}}, “type”: “function”}
</tools>

For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{“name”: <function-name>, “arguments”: <args-json-object>}
</tool_call>< |im_end| >
< |im_start| >user
Can you fetch some random advice for me?< |im_end| >
< |im_start| >assistant

Table 12: An example for tool use of GPT series models. The tools are send with the ‘tools’ key of ‘OpenAI().chat.completions.create()’.

Can you fetch some random advice for me?

Table 13: An example for tool use of TL-CodeLLaMA-2.

System: Function:
def random_advice():
“““
Returns a random advice slip as a slip object.

Args:
”””

User: Can you fetch some random advice for me?
Assistant:

E Examples of Tool Call and Feedback

In Table 14, we present some examples of potential scenarios that may arise during tool calls.

Table 14: Examples of various scenarios that may arise during tool calls.

Category	Models	Output	Feedback
All Right	TL-CodeLLaMA-2	random_advice()	{"slip": {"id": 52, "advice": "Don't promise what you can't deliver."}}
Tool Instability	GPT-4-turbo	{"arguments": "{}", "name": "get_currency"}	HTTPSConnectionPool(host=..., port=443): Max retries exceeded with url: ...
Tool Instability	ChatGLM-4-chat	get_exchange_rate {"from_currency": "EUR", "to_currency": "GBP"}	Invalid API call. Currency codes might be invalid.
Tool Instability	Qwen-2-Instruct	Thought: ... Action: google_trends_search Action Input: {"query": "school", "data_type": "related_topics"}	400 Client Error: Bad Request for url: ...
Tool Call Failure	ChatGLM-4-chat	search_country {"query": "IN"}	{"error": "Response error."}
Tool Call Failure	GPT-3.5-turbo	{"arguments": "{"query": "Bitcoin", "data_type": "RELATED_QUERIES", "limit": "5"}", "name": "google_trends_search"}	'<=' not supported between instances of 'str' and 'int'
Tool Call Failure	ToolLLaMA-2	Thought: ... Action: get_nobel_results Action Input: { "year": "2018" }	Object of type bytes is not JSON serializable
Tool Hallucination	Qwen-2.5-Instruct	{"arguments": "{"query": "most popular tourist destinations in Europe"}", "name": "google_search"}	name google_search if not defined
Parameter Hallucination	ToolLLaMA-2	Thought: ... Action: get_news_headlines Action Input: { "api_key": "your_api_key", "q": "technology", "sortBy": "popularity"}	get_news_headlines() got an unexpected keyword argument 'sortBy'
Parameter Missing	Qwen-2.5-Instruct	{"arguments": "{"q": "London", "days": "7", "api_key": "your_api_key"}", "name": "forecast"}	forecast() missing 1 required positional argument: aqi
Others	GPT-4o	{"arguments": "{"website": "www.mywebsite.com"}", "name": "analyze_scan"}	{"error": "Recently completed scan for www.mywebsite.com not found"}